The Intel® C/C++ Compiler

The Intel® C/C++ Compiler V4.0 enables superior performance by providing processor-specific optimizations for the Intel® Pentium® III processor.

Feature Summary

- Compatibility with Microsoft* Visual C++*
- Plug-in to the Microsoft Visual Studio*
- Optimizations for the latest Intel® Architecture processors
- Streaming SIMD Extensions and MMX™ instructions to enable optimized multimedia applications
- C++ class libraries to facilitate implementation of Streaming SIMD Extensions and MMX technology intrinsics
- Interprocedural optimizations (IPOs)
- Profile-guided optimizations (PGOs)
- Vectorization options
- Excellent floating-point instruction scheduling

Compatible with Visual C++* as a Plug-in to Visual Studio*

The Intel C/C++ Compiler v4.0 plugs into the Microsoft Visual Studio, and is compatible with source code written for versions 4.2, 5.0, and 6.0 of the Microsoft Visual C++ compiler as follows:

- Compilation switches and optimizations
- Makefile support
- Inline assembly language
- Object module, library, and DLL formats
- Debug and symbol formats

You can port your code from Visual C/C++ without modifications and use the Intel C/C++ compiler from within the Visual Studio. You have the flexibility of compiling one or all of the modules of an existing project with the Intel C/C++ Compiler. This provides the opportunity to recompile modules where you stand to gain performance improvements by using the Intel C/C++ Compiler and link them with other modules compiled with the Visual C/C++ compiler.

If you already have Microsoft Visual C/C++ 4.2 or higher on your system, the installation procedure automatically installs the Intel compiler selection tool, from which you can select the Intel C/C++ Compiler.

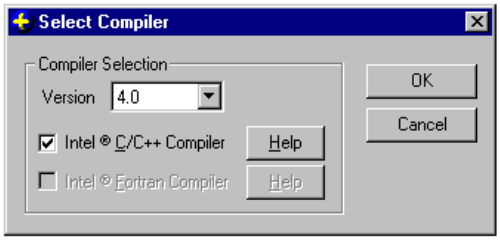


Figure 1: Compiler Selection Tool

The Streaming SIMD Extensions and MMX™ Technology

A key feature of the Intel C/C++ Compiler is its support for application development with the Intel Streaming SIMD Extensions and MMX™ instructions. These extensions provide the power to develop optimized code for multimedia applications.

The Streaming SIMD Extensions consist of about seventy instructions used primarily for 3D geometry, imaging, video, and speech recognition. The MMX instruction set comprises another fifty-seven instructions designed to efficiently manipulate and process video, audio, and graphical data.

In addition to the potential performance gains, the Intel C/C++ compiler simplifies the development process for you by providing four methods of implementing these powerful instructions (see **Figure 2**):

- *Inline Assembly Language*—With the `__asm` keyword, you can directly insert assembly code into your source code.
- *C style programming using intriniscs*—The Intel compiler manages the

Inline Assembly:

```
__m128 a,b,c;
     movaps xmm0,b
     movaps xmm1,c
/* Add the four single-precision floating-point values of xmm0 and
xmm1*/
     addps xmm0,xmm1
     movaps a, xmm0
```

Intrinsics Using C-Style Syntax:

```
__m128 a,b,c;
/* Add the four single-precision floating-point values of b and c */
       a = _mm_add_ps(b,c)
```

C++ Libraries:

```
F32vec4 a,b,c;
/* Add the four single-precision floating-point values of b and c */
     a = b + c
```

Vectorization:

```
float a[4], b[4], c[4];
int i;

#pragma vector
    for (i=0; i < 4; i++)
/* Add the four single-precision floating-point values of b and c */
       a[i] = b[i] + c[i];
    ...
```

Figure 2. The Intel Compiler's Three Methods of Using Extensions

registers and the loads and stores to schedule the key instructions that take advantage of the Streaming SIMD Extensions and MMX technology intrinsics.

- *C++ class libraries*—Intel has provided a library of C++ classes for both Streaming SIMD Extensions and MMX technology intrinsics using C++ operator overloading. These classes enable manipulation of SIMD objects through the use of familiar operators.
- *Automatic Vectorization*—The Intel C/C++ Compiler provides a vectorizer to automatically parallelize code to take maximum advantage of the underlying processor capabilities. The vectorizer's options are documented with examples to demonstrate how you can increase the speed of your application execution. See the *Intel C/C++ Compiler User's Guide* for detailed descriptions of the vectorizer, its options, and example usage.

Figure 3 compares the performance vs portability aspects of using the different coding methods.

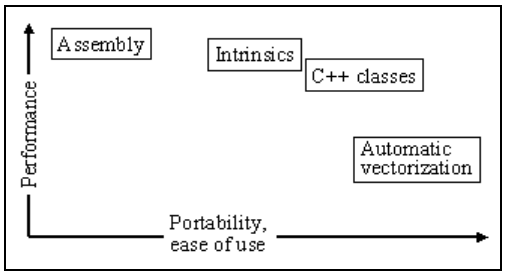


Figure 3. Performance vs Portability

Profile-Guided Optimization (PGO)

PGO enables the compiler to focus its efforts more effectively on regions of the code that affect your program at runtime. PGO also enables the compiler to optimize the following:

- hardware resources on the processor
- instruction cache and paging
- effectiveness of branch prediction

With PGO, the Intel C/C++ Compiler

creates an instrumented program from your source. Each time you execute the instrumented code, the compiler generates a dynamic information file. When you compile again, the dynamic information files that have already been created are merged into a summary file.

Using the profile information in the summary file, the compiler optimizes the execution of the most heavily travelled paths in your program. For example, PGO might find that a certain section of code is rarely executed. This code is then moved to the end of the module, resulting in more efficient processor-instruction fetching.

The following are the phases of PGO, which when completed, provide the data that can significantly improve performance of large applications (see Figure 4):

Phase 1—Instrumented Compilation: The compiler inserts instrumentation code into your program to produce a profile.

Phase 2:—Instrumented Execution: Each time you execute your program, the compiler creates a dynamic information file that represents your program's actual execution behavior for that run.

Phase—3: Feedback Compilation When you compile again, the compiler uses the dynamic information files from all previous runs to optimize to the profile of your program and produces a dynamic information summary file**.**

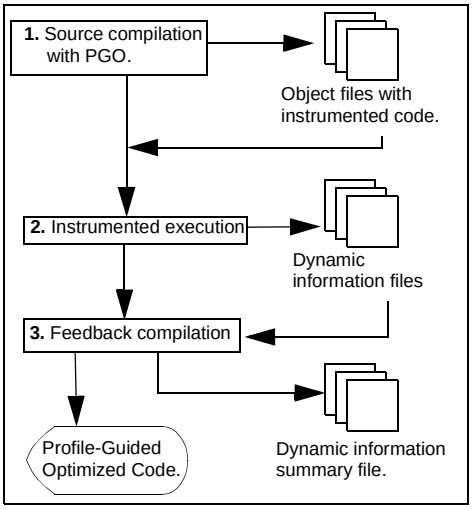


Figure 4. Three Phases of PGO

Through PGO, the Intel C/C++ Compiler has been proven to dramatically assist applications such as large databases. PGO can help eliminate instruction-cache thrashing by reorganizing code layout, shrinking code size, and reducing branch mispredictions.

Interprocedural Optimization (IPO)

The Intel C/C++ Compiler provides analysis for interprocedural optimizations to assist you with programs that contain many small or medium-sized frequently used functions, especially for programs that contain calls within loops.

Potential optimizations around calling points are normally inhibited due to lack of information about what happens in the called procedure. Interprocedural analysis examines the relationship between calling and called procedures and enables the following optimizations:

- Function inlining
- Passing arguments in registers
- Interprocedural constant propagation

Inline function expansion is one of the main optimizations performed by the inter-procedural optimizer. For function calls that the compiler believes are frequently executed, the compiler can replace the instructions of the call with code for the function itself.

Further, the Intel C/C++ compiler can perform inline function expansion for calls between procedures defined in separate files. Multifile IPO performs different work depending on whether compilation or linkage is being performed.

- *Compilation Phase*—As each source file is compiled, multifile IPO creates an information file with a `.il` suffix, which contains an intermediate representation of the source code and summary information used for optimization.
- *Linkage Phase*—When you specify multifile IPO, the compiler is invoked again and performs IPO across all modules for which a corresponding up-to-date `.il` file exists.

Excellent floating-point instruction throughput

The C/C++ Compiler exploits the use of the stack for efficient executions of floating-point (FP) instructions.

Most FP operations have a dependence on the previous instruction, which requires that one operand and the result use the top of the stack. The Intel C/C++ Compiler breaks this dependency by allowing inputs for the next instructions to always be at the top of the stack.The compiler provides this capability by effective use of the FXCH instruction, which comes at almost no additional cost on the Pentium® Pro and later generation processors.

Consider the following expression:

```
a = ((b + c) * b) + ((d + e) * d);
```

This expression can be presented graphically as shown in Figure 8. The serial instruction sequence depicts instructions executed one at a time with no overlapping because of the top-of-stack dependency.

The parallel instruction sequence uses the FXCH instruction that provides the following gains:

- Overlapping instructions that can put their calculation results in any stack register, not necessarily to the top of the stack, but different stack registers
- More parallelism achieved

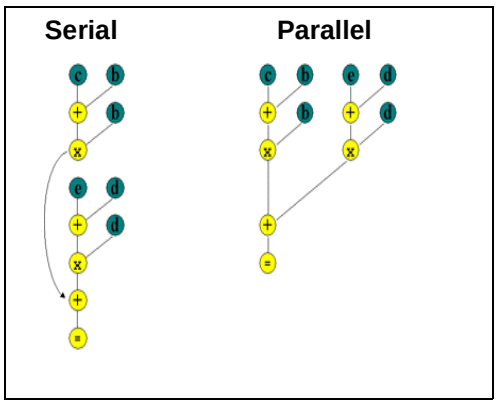


Figure 5. Serial and Parallel Sequencing

Additional Features

Intel's C/C++ Compiler V4.0 provides a comprehensive set of general-purpose optimizations and features for use with a broad range of applications. They include those listed in Table 1 and Table 2.

Table 2: Intel® C/C++ Compiler Optimization List:

Optimization	Affected Aspect of the Program
optimized code selection	instruction selection/ addressing modes
global register allocation	register use
instruction scheduling	instruction reordering
register variable detection	register use
common subexpression	constants and expression evaluation elimination
dead-code elimination	instruction sequencing
variable renaming	register use
loop-invariant code movement	instruction sequencing
copy propagation	constants and expression evaluation
constant propagation	constants and expression evaluation
strength reduction/induction variable simplification	instruction selection/ sequencing, constants and expression evaluation
inline function expansion	calls, jumps, branches, and loops
interprocedural constant propagation	arguments, global variables, and return values
passing arguments in registers	calls, register usage
monitoring module-level static variables	further optimizations, loop invariant code
multifile optimization	performs optimizations across files
loop unrolling	converts loops into sequentially executed statements

Table 1: Additional Benefits and Features of the Intel® C/C++ Compiler

Feature Provided	Benefit
Alternate tool selection for preprocessing and compilation.	Provides a flexible software development environment.
Comprehensive use of addressing modes.	Reduces the number of instructions in program.
Whole-program compilation.	Enables multifile optimizations.
Processor dispatching support.	Allows versions of code for different Intel® processors to exist in a single binary and then optimizes by using the code for the actual processor on which the application is running.
Provides processor-specific optimizations.	Allows you to maximize performance on a specific Intel Architecture processor, including the Pentium® III processor.

Information in this document is provided in connection with Intel products. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document. Except as provided in Intel's Terms and Conditions of Sale for such products, Intel assumes no liability whatsoever, and Intel disclaims any express or implied warranty, relating to sale and/or use of Intel products including liability or warranties relating to fitness for a particular purpose, merchantability, or infringement of any patent, copyright or other intellectual property right. Intel products are not intended for use in medical, life saving, or life sustaining applications. Intel may make changes to specifications and product descriptions at any time, without notice.

*Third-party brands and names are the property of their respective owners.

© 1999 Intel Corporation

Printed in USA/2K/0598/JM, Order Number: 244461-001